Unpacking Colibri Loader: A Russian APT linked Campaign

bitsight.com/blog/unpacking-colibri-loader-russian-apt-linked-campaign

Written by André Tavares November 30, 2022 Share Facebook Twitter LinkedIn



Between July and October 2022 BitSight observed a ColibriLoader malware campaign being distributed by PrivateLoader, which was identified as being utilized by the threat actor UAC-0113, a group linked to Sandworm by CERT-UA. <u>Sandworm</u> is known to be a Russian advanced persistent threat (APT) group affiliated with The Main Directorate of the General Staff of the Armed Forces of the Russian Federation (GRU). In this research, we present how to manually "unpack" a sample from a recent campaign. Unpacking means reaching the final stage of the malware, which contains its main functionality. We also share some threat-hunting signatures and indicators of compromise which can be utilized in defense and tracking efforts.

About Colibri

ColibriLoader is a Malware-as-a-Service family, first advertised on XSS.is <u>cybercrime forum</u> <u>in August 2021</u> to "people who have large volumes of traffic and lack of time to work out the material" (Fig. 1). For \$150/week or \$400/month, it offers a small, unpacked, obfuscated loader written in C and assembly, along with a control panel written in PHP. As its name suggests, it's meant to deliver and manage payloads onto infected computers. Moreover, <u>the</u> malware ignores systems from Commonwealth of Independent States countries (Armenian, Azerbaijani, Belarusian, Hungarian, Kazakh, Kyrgyz, Romanian, Russian, Tajik, Turkmen, Uzbek).

2 3 Forward >		Go to new T	Trac
423	27.08.2021	α ₀ 0 []	#
	I present to your attention my development - Colibri Loader. The software is designed to deliver and run your executable files and dll libraries of Ideal for people with large volumes of traffic and lack of time to study the material.	on infected PCs.	
Th_tha Premium	The bot is written in C / ASM (crt, stl are not used), works on Windows 7/8/10/11 and server counterparts, without any dependencies . Easy not contain resources, IAT, TLS, only relocks and code section). All internal string literals are encrypted, traffic between the bot and the C&C se encrypted. Supplied in two formats:)es
egistration : 01.04.2020 Posts : 334 Reactions : 311 Deal guarantor : 1	 .exe - weight ~ 20kb, does not require any additional manipulations with the crypt. .dll - weight ~ 17kb. Not installed on the system! Running through the exported function. 		
Deposit : 0.0086 B	Bot functionality:		
	• Run .exe from user or from admin (runas + cmd), support for launch arguments		
	Running x86 .exe / .dll in memory via LoadPE Running x86 .dll via rundll32		
	Running x86 .dll via LoadLibrary		
	Running x86 .dll via regsrv32		
	Executing cmd commands Updating the bot with a fresh crypt or a new version		
	Removing a bot from an infected device.		

Fig. 1 - Post on XSS.is cybercrime forum by user "c0d3r_0f_shr0d13ng3r"

PrivateLoader distributing Packed Colibri

<u>PrivateLoader</u> is a loader from a pay-per-install malware distribution service that has been utilized to distribute info stealers, banking trojans, loaders, spambots, rats, miners and ransomware on Windows machines. While monitoring PrivateLoader malware distribution activity, we spotted ColibriLoader being distributed between <u>July and October</u>. Many security products automatically classified these samples and we noticed that all of them have the tag "<u>Build1</u>", which may represent the botnet or campaign id. Contradicting the author's advertisements, we noticed some indicators suggesting that the samples are packed, such as their size, which should be around 20KB, and the fact that it contains only two sections, the *.text* and *.reloc*, which was not the case.

In order to evade antivirus security products and frustrate malware reverse engineering, malware operators leverage encryption and compression via executable packing to protect their malicious code. Malware packers are software programs that either encrypt or compress the original binary, making it unreadable until it's placed in memory. In general, malware packers consist of two components, a packed buffer, the actual malicious code, and an unpacking stub responsible for unpacking and executing the packed buffer. Threat Actors make use of packers when distributing their malware as they remain an effective way to evade detection and make the malware harder to analyze. Manual analysis can defeat these protections and help develop tools that aid in this costly task.

Unpacking ColibriLoader

Let's have a look at a <u>sample</u> dropped by PrivateLoader on September 4, 2022. First, we tried to use <u>unpac.me</u> service to try to unpack it automatically but we were unlucky. So, let's dive deep into this sample.

Resolving the Windows API

Opening it on IDA Pro, on the main function (Fig. 3), we can see a pattern that seems to be a way of dynamically resolving some Windows API functions, which are usually crucial to understand the code behavior quickly. The malware walks the process environment block (PEB), looking for the in-memory loaded modules base addresses on the current process, finds the functions exported by each, hashes them, and compares it with the hash of LoadLibrary. Then, it calls *LoadLibraryA* to load *kernel32.dll* to get a module handle for it, but does not actually load it since it is already loaded by default, and then searches for the target export function (a more detailed explanation can be found <u>here</u>). By searching on Google for the constants in the code that generates the hash, we confirmed the hashing algorithm in use is <u>Fowler–Noll–Vo</u>.

```
for ( i = NtCurrentTeb()->ProcessEnvironmentBlock->Ldr->InLoadOrderModuleList.Flink; ; i = v87->Flink )
  {
   Flink = i[3].Flink;
   v87 = i;
   v5 = (Flink + *(&Flink[7].Blink[15].Flink + Flink));
   if ( v5 != Flink )
   -{
     v6 = v5[3].Flink == 0;
     v89 = 0;
     if ( !v6 )
       break;
   3
LABEL_11:
   ;
 }
 v7 = (&Flink->Flink + v5[4].Flink);
 while (1)
 {
   v8 = Flink + *v7;
   v90 = 0x811C9DC5;
                                              // Fowler-Noll-Vo offset basis
   v91 = *v8;
   v9 = v89;
   if ( v91 )
     break;
LABEL_10:
   ++v7:
   v89 = v9 + 1;
   if ( (v9 + 1) >= v5[3].Flink )
     goto LABEL_11;
 -}
 v10 = v91;
 v11 = v90;
 do
 {
   v11 = 0x1000193 * (v11 ^ v10);
                                              // Fowler-Noll-Vo prime
   v10 = *++v8;
 while ( *v8 );
 v90 = v11;
  v12 = 0;
 if ( v90 != 0x53B2070F )
                                              // LoadLibraryA
 {
   v9 = v89;
   goto LABEL_10;
  v89 = ((Flink + *(&v5[3].Blink->Flink + 4 * *(&v5[4].Blink->Flink + 2 * v89 + Flink) + Flink)))("kernel32.dll");
```

Fig. 3 - Example of Windows API resolution.

Encrypted Shellcode and Executable

Looking a bit further through the code, we can spot what seems to be an XOR decryption of 520 bytes of shellcode at 0x454708, where the key is "2760", and also the change in the protection of that region (*VirtualProtect*) to PAGE_EXECUTE_READWRITE (0x40), as well as four calls of a function within that region (Fig. 4).

```
for (idx = 0; idx < 520; ++idx)
 shellcode_454708[idx] ^= xor_key_1[idx & 3];// XOR decrypt
v43 = (v36 + *(*(v36 + 60) + v36 + 120));
for ( ii = (v36 + v43[8]); ; ++ii )
{
 v45 = (v36 + *ii);
 v92 = 0x811C9DC5;
 v46 = *v45;
 if ( *v45 )
  {
   v47 = v92;
   do
   {
     v47 = 0 \times 1000193 * (v47 ^ v46);
     v46 = *++v45;
   }
   while ( *v45 );
   v92 = v47;
                                            // VirtualProtect
   v6 = v47 == 0x820621F3;
   v36 = hKerne132;
   if ( v6 )
     break;
 }
 ++v41;
}
((hKernel32 + *(v43[7] + 4 * *(v43[9] + 2 * v41 + hKernel32) + hKernel32)))(shellcode_454708, 520, 0x40, v82);
v48 = (shellcode_4547A8)(&unk_4C2508, 1090, 5, &v86, &ntdll_dll, &RtlAllocateHeap);
v81 = v48;
v88 = (shellcode_4547A8)(&unk_454910, 449528, 10, &v83, &ntdll_dll, &RtlAllocateHeap);
v90 = (shellcode_4547A8)(&unk_4C2950, 153600, 10, &v85, &ntdll_dll, &RtlAllocateHeap);
v87 = (shellcode_4547A8)(&unk_454020, 1761, 5, &v84, &ntdll_dll, &RtlAllocateHeap);
```

Fig. 4 - Shellcode decryption.

We can confirm it by disassembling the shellcode function at 0x4547A8 after running it on x32dbg (Fig. 5)

O04547AB 55 push ebp 004547AB 88EC mov ebp,esp 004547AB 83EC 10 sub esp,10 004547AE 53 push ebx 004547AF 56 push esi 004547AF 56 push esi 004547AF 68 271B595E push 5E591B27 00454780 57 push 9840DD03 00454786 E8 4DFFFFFF call colibri_04_09_2022.454708 00454700 8BF0 mov esi, eax 004547C2 E8 41FFFFFF call colibri_04_09_2022.454708 004547C2 E8 35FFFFFF call colibri_04_09_2022.454708 004547C2 8BF8 mov edi, eax 004547C2 E8 35FFFFFF call colibri_04_09_2022.454708 004547D8 8JE0 mov edi, eax 004547D6 FFD0 call colibri_04_09_2022.454708 004547D8 BBD8 mov edi, eax 004547D8 BBD8 mov edi, eax 004547D8 BBD8 mov ebx,eax 004547D8 BSD FC push dword ptr ss:[ebp+4],ebx<			
004547AB 83EC 10 sub esp,10 004547AE 53 push ebx 004547AF 56 push esi 004547B0 57 push edi 004547B1 68 271B595E push 5E591B27 004547B6 E8 4DFFFFFF call colibri_04_09_2022.454708 004547C0 8BF0 mov esi,eax 004547C2 E8 41FFFFFF call colibri_04_09_2022.454708 004547C6 8BF0 mov esi,eax 004547C7 68 D3ACAC1D push 1DACACD3 004547C8 BF8 mov edi,eax 004547C6 E8 35FFFFFF call colibri_04_09_2022.454708 004547C6 BBF8 mov edi,eax 00454705 FFD0 call eax 00454706 FFD0 call eax 00454708 8BD8 mov ebx,eax 00454709 895D FC mov dword ptr ss:[ebp+18] 00454720 FFD6 call esi 00454725 50 push dword ptr ss:[ebp+1C] 00454726 FFD7 call edi 00454728 877 push edi 00454728 57 <th>004547A8</th> <th>55</th> <th></th>	004547A8	55	
004547AE 53 push ebx 004547AF 56 push esi 004547AF 56 push edi 004547B0 57 push sei 004547B1 68 271B595E push sei 004547B6 E8 4DFFFFF call colibri_04_09_2022.454708 004547C0 8BF0 mov esi,eax 004547C2 E8 41FFFFFF call colibri_04_09_2022.454708 004547C2 E8 41FFFFFF call colibri_04_09_2022.454708 004547C2 B8F8 mov esi,eax 004547C2 B8F8 mov edi,eax 004547C2 B8F8 mov edi,eax 004547C2 B8F8 mov edi,eax 004547C3 83C4 OC add esp,C 004547D8 FF75 18 push dword ptr ss:[ebp+18] 004547D8 8BD8 mov ebx,eax 004547E0 FFD6 call esi 004547E4 FF75 1C push dword ptr ss:[ebp+4],ebx 004547E5 50 push eax 004547E6 FFD7 call edi 004547E8 8B7D 0C mov edi,dword ptr ss:[ebp+C] 004547E6	004547A9	8BEC	
004547AF 56 push esi 00454780 57 push edi 00454781 68 271B595E push 5E591B27 00454786 E8 4DFFFFFF call colibri_04_09_2022.454708 00454700 8BF0 mov esi,eax 00454701 8BF0 mov esi,eax 00454702 E8 41FFFFFF call colibri_04_09_2022.454708 00454703 8BF0 mov esi,eax 00454704 8BF8 mov edi,eax 00454705 E8 35FFFFFF call colibri_04_09_2022.454708 00454706 FFD0 call colibri_04_09_2022.454708 00454705 8BF8 mov edi,eax 00454706 FFD0 call colibri_04_09_2022.454708 00454706 FFD0 call colibri_04_09_2022.454708 00454708 8B78 mov edi,eax 00454708 BSD8 mov edi,eax 00454709 895D FC mov dword ptr ss:[ebp+18] 00454720 FFD6 call esi 00454725 50 push dword ptr ss:[ebp+10] 00454726 FFD7 call edi 00454728 8B7D 0C mov edi,dw	004547AB	83EC 10	sub esp,10
00454780 57 push edi 00454781 68 271B595E push 5E591B27 00454786 E8 4DFFFFFF call colibri_04_09_2022.454708 00454788 68 03DD4098 push 9840DD03 00454700 8BF0 mov esi,eax 00454702 E8 41FFFFFF call colibri_04_09_2022.454708 00454703 8BF0 mov esi,eax 00454704 8BF8 mov edi,eax 00454705 E8 35FFFFFF call colibri_04_09_2022.454708 00454706 FFD0 gas 00454707 68 D3ACAC1D push 1DACACD3 00454705 B8F8 mov edi,eax 00454706 FFD0 call colibri_04_09_2022.454708 00454708 BF75 18 push dword ptr ss:[ebp+18] 00454708 FF75 18 push dword ptr ss:[ebp+4],ebx 00454709 S95D FC mov dword ptr ss:[ebp+4],ebx 0045472 FF75 1C push dword ptr ss:[ebp+1C] 00454725 S0 push eax 00454726 FFD7 call edi 00454728 887D 0C mov edi,dword ptr ss:[ebp+4],eax 004547E5 <td>004547AE</td> <td>53</td> <td>push ebx</td>	004547AE	53	push ebx
00454781 68 2718595E push 5E591827 00454786 E8 4DFFFFFF call colibri_04_09_2022.454708 00454788 68 03DD4098 push 9840DD03 00454700 8BF0 mov esi,eax 00454702 E8 41FFFFFF call colibri_04_09_2022.454708 00454702 E8 41FFFFFF call colibri_04_09_2022.454708 00454702 B8F8 mov esi,eax 00454702 B8F8 mov edi,eax 00454703 83C4 0C add esp,C 00454706 FFD0 call eax 00454708 FF75 18 push dword ptr ss:[ebp+18] 00454708 8BD8 mov ebx,eax 00454709 895D FC push dword ptr ss:[ebp+4],ebx 00454720 FFD6 call edi 00454725 50 push dword ptr ss:[ebp+1C] 00454726 FFD7 call edi 00454727 6A 08 push edi 00454728 870 0C mov edi,dword ptr ss:[ebp+6],eax 00454725 53 push ebx 00454726 FA S3 00454727 6A 08 push ebx	004547AF	56	push esi
00454786 E8 4DFFFFFF call colibri_04_09_2022.454708 00454788 68 03DD4098 push 9840DD03 00454700 8BF0 mov esi,eax 00454702 E8 41FFFFFF call colibri_04_09_2022.454708 00454702 E8 41FFFFFF call colibri_04_09_2022.454708 00454702 E8 41FFFFFF call colibri_04_09_2022.454708 00454702 8BF8 mov edi,eax 00454702 E8 35FFFFFF call colibri_04_09_2022.454708 00454705 FFD6 call eax 00454706 FFD0 call eax 00454708 SBD8 mov ebx,eax 00454708 BBD8 mov dword ptr ss:[ebp+18] 00454709 895D FC mov dword ptr ss:[ebp+4],ebx 00454720 FFD6 call esi 00454725 S0 push dword ptr ss:[ebp+1C] 00454726 FFD7 call edi 00454728 887D 0C mov edi,dword ptr ss:[ebp+C] 00454726 GA 08 push edi 00454727 6A 08 push ebx 00454728 S945 F8 mov dword ptr ss:[ebp-8],eax	004547B0	57	
00454788 68 03DD4098 push 9840DD03 004547C0 88F0 mov esi,eax 004547C2 E8 41FFFFFF call colibri_04_09_2022.454708 004547C7 68 D3ACAC1D push 1DACACD3 004547C2 88F8 mov edi,eax 004547C2 E8 35FFFFFF call colibri_04_09_2022.454708 004547C2 88F8 mov edi,eax 004547C2 E8 35FFFFFF call colibri_04_09_2022.454708 004547D3 83C4 0C add esp,C 004547D6 FFD0 call eax 004547D8 8F75 18 push dword ptr ss:[ebp+18] 004547D8 8BD8 mov ebx,eax 004547D8 8BD8 mov dword ptr ss:[ebp-4],ebx 004547E0 FFD6 call esi 004547E1 FF75 1C push dword ptr ss:[ebp+1C] 004547E2 FF75 1C push eax 004547E3 S0 push eax 004547E4 FFD7 call edi 004547E5 S0 push edi 004547E8 887D 0C mov edi,dword ptr ss:[ebp+6],eax 004547E4 S3 push ebx	004547B1	68 271B595E	push 5E591B27
004547C0 88F0 mov esi,eax 004547C2 E8 41FFFFFF call colibri_04_09_2022.454708 004547C7 68 D3ACAC1D push 1DACACD3 004547C2 88F8 mov edi,eax 004547C2 E8 35FFFFFF call colibri_04_09_2022.454708 004547C2 88F8 mov edi,eax 004547C2 E8 35FFFFFF call colibri_04_09_2022.454708 004547D3 83C4 0C add esp,C 004547D6 FFD0 call eax 004547D8 FF75 18 push dword ptr ss:[ebp+18] 004547D8 88D8 mov ebx,eax 004547D0 895D FC mov dword ptr ss:[ebp+4],ebx 004547E2 FF75 1C push dword ptr ss:[ebp+1C] 004547E5 50 push eax 004547E6 FFD7 call edi 004547E8 887D 0C mov edi,dword ptr ss:[ebp+C] 004547E4 53 push edi 004547E5 53 push ebx 004547E5 53 push ebx 004547E5 53 push ebx 004547E5 53 push ebx <	004547B6	E8 4DFFFFFF	call colibri_04_09_2022.454708
004547C2 E8 41FFFFFF call colibri_04_09_2022.454708 004547C7 68 D3ACAC1D push 1DACACD3 004547C2 8BF8 mov edi,eax 004547C2 E8 35FFFFFF call colibri_04_09_2022.454708 004547C2 8BF8 mov edi,eax 004547C2 E8 35FFFFFF call colibri_04_09_2022.454708 004547D2 83C4 0C add esp,C 004547D6 FFD0 call eax 004547D8 FF75 18 push dword ptr ss:[ebp+18] 004547D0 895D FC mov ebx,eax 004547E2 FF75 1C push dword ptr ss:[ebp+4],ebx 004547E5 50 push eax 004547E6 FFD7 call edi 004547E8 887D 0C mov edi,dword ptr ss:[ebp+C] 004547E4 57 push edi 004547E5 53 push ebx 004547E5 53 push ebx 004547E5 53 push ebx 004547E5 53 push ebx	004547BB	68 03DD4098	push 9840DD03
004547C7 68 D3ACAC1D push 1DACACD3 004547CC 88F8 mov edi,eax 004547CE E8 35FFFFFF call colibri_04_09_2022.454708 004547D3 83C4 0C add esp,C 004547D6 FFD0 call eax 004547D8 FF75 18 push dword ptr ss:[ebp+18] 004547DB 8BD8 mov ebx,eax 004547DD 895D FC mov dword ptr ss:[ebp-4],ebx 004547E2 FF75 1C push dword ptr ss:[ebp+1C] 004547E3 50 push eax 004547E4 FFD7 call edi 004547E5 50 push eax 004547E8 8B7D 0C mov edi,dword ptr ss:[ebp+C] 004547E4 57 push edi 004547E5 53 push ebx 004547E5 53 push ebx 004547E5 53 push ebx	004547C0	8BF0	mov esi,eax
004547CC 88F8 mov edi,eax 004547CE E8 35FFFFFF call colibri_04_09_2022.454708 004547D3 83C4 0C add esp,C 004547D6 FFD0 call eax 004547D8 FF75 18 push dword ptr ss:[ebp+18] 004547D8 88D8 mov ebx,eax 004547D0 895D FC mov dword ptr ss:[ebp-4],ebx 004547E2 FF75 1C push dword ptr ss:[ebp+1C] 004547E5 50 push eax 004547E6 FFD7 call edi 004547E8 887D 0C mov edi,dword ptr ss:[ebp+C] 004547E5 53 push ebx 004547E5 53 push ebx 004547E6 FF8 mov dword ptr ss:[ebp-8],eax	004547C2	E8 41FFFFFF	call colibri_04_09_2022.454708
004547CE E8 35FFFFFF call colibri_04_09_2022.454708 004547D3 83C4 0C add esp,C 004547D6 FFD0 call eax 004547D8 FF75 18 push dword ptr ss:[ebp+18] 004547D8 88D8 mov ebx,eax 004547D0 895D FC mov dword ptr ss:[ebp-4],ebx 004547E0 FFD6 call esi 004547E5 50 push dword ptr ss:[ebp+1C] 004547E6 FFD7 call edi 004547E8 887D 0C mov edi,dword ptr ss:[ebp+C] 004547E6 57 push edi 004547E6 53 push 8 004547E6 53 push 8 004547E7 53 push ebx 004547E7 53 push ebx	004547C7	68 D3ACAC1D	push 1DACACD3
004547D3 83C4 OC add esp,C 004547D6 FFD0 call eax 004547D8 FF75 18 push dword ptr ss:[ebp+18] 004547D8 88D8 mov ebx,eax 004547D0 895D FC mov dword ptr ss:[ebp-4],ebx 004547E0 FFD6 call esi 004547E2 FF75 1C push dword ptr ss:[ebp+1C] 004547E5 50 push eax 004547E6 FFD7 call edi 004547E8 887D OC mov edi,dword ptr ss:[ebp+C] 004547E6 FFD7 push edi 004547E8 57 push edi 004547E6 FA push edi 004547E6 53 push ebx 004547E6 53 push ebx 004547EF 8945 F8 mov dword ptr ss:[ebp-8],eax	004547CC	8BF8	mov edi,eax
O04547D6 FFD0 call eax O04547D8 FF75 18 push dword ptr ss:[ebp+18] O04547D8 8BD8 mov ebx,eax O04547D0 895D FC mov dword ptr ss:[ebp-4],ebx O04547E0 FFD6 call esi O04547E2 FF75 1C push dword ptr ss:[ebp+1C] O04547E6 FFD7 call edi O04547E8 887D OC mov edi,dword ptr ss:[ebp+C] O04547E6 FFD7 push edi O04547E8 57 push edi O04547E6 57 push edi O04547E6 53 push ebx O04547EF 53 push ebx	004547CE	E8 35FFFFFF	call colibri_04_09_2022.454708
004547D8 FF75 18 push dword ptr ss:[ebp+18] 004547DB 8BD8 mov ebx,eax 004547DD 895D FC mov dword ptr ss:[ebp-4],ebx 004547E0 FFD6 call esi 004547E2 FF75 1C push dword ptr ss:[ebp+1C] 004547E5 50 push eax 004547E6 FFD7 call edi 004547E8 8B7D 0C mov edi,dword ptr ss:[ebp+C] 004547E8 57 push edi 004547E2 6A 08 push 8 004547E5 53 push ebx 004547EF 8945 F8 mov dword ptr ss:[ebp-8],eax	004547D3	83C4 0C	add esp,C
004547DB 8BD8 mov ebx,eax 004547DD 895D FC mov dword ptr ss:[ebp-4],ebx 004547E0 FFD6 call esi 004547E2 FF75 1C push dword ptr ss:[ebp+1C] 004547E5 50 push eax 004547E6 FFD7 call edi 004547E8 887D 0C mov edi,dword ptr ss:[ebp+C] 004547E6 57 push edi 004547E6 57 push edi 004547E5 53 push ebx 004547EF 8945 F8 mov dword ptr ss:[ebp-8],eax	004547D6	FFD0	call eax
004547DD 895D FC mov dword ptr ss:[ebp-4],ebx 004547E0 FFD6 call esi 004547E2 FF75 1C push dword ptr ss:[ebp+1C] 004547E5 50 push eax 004547E6 FFD7 call edi 004547E8 887D 0C mov edi,dword ptr ss:[ebp+C] 004547E6 57 push edi 004547E8 57 push edi 004547E5 53 push ebx 004547EF 8945 F8 mov dword ptr ss:[ebp-8],eax	004547D8	FF75 18	push dword ptr ss:[ebp+18]
004547E0 FFD6 call esi 004547E2 FF75 1C push dword ptr ss:[ebp+1C] 004547E5 50 push eax 004547E6 FFD7 call edi 004547E8 8B7D 0C mov edi,dword ptr ss:[ebp+C] 004547E8 57 push edi 004547E2 6A 08 push 8 004547E5 53 push ebx 004547EF 8945 F8 mov dword ptr ss:[ebp-8],eax	004547DB	8BD8	mov ebx,eax
004547E2 FF75 1C push dword ptr ss:[ebp+1C] 004547E5 50 push eax 004547E6 FFD7 call edi 004547E8 887D 0C mov edi,dword ptr ss:[ebp+C] 004547E8 57 push edi 004547EC 6A 08 push ebx 004547EF 53 push ebx 004547EF 8945 F8 mov dword ptr ss:[ebp-8],eax	004547DD	895D FC	mov dword ptr ss:[ebp-4],ebx
004547E5 50 push eax 004547E6 FFD7 call edi 004547E8 8B7D OC mov edi,dword ptr ss:[ebp+C] 004547EB 57 push edi 004547EC 6A 08 push ebx 004547EF 53 push ebx 004547EF 8945 F8 mov dword ptr ss:[ebp-8],eax	004547E0	FFD6	call esi
004547E6 FFD7 call edi 004547E8 8B7D OC mov edi,dword ptr ss:[ebp+C] 004547EB 57 push edi 004547EC 6A 08 push ebx 004547EF 53 push ebx 004547EF 8945 F8 mov dword ptr ss:[ebp-8],eax	004547E2	FF75 1C	push dword ptr ss:[ebp+1C]
004547E8 8B7D OC mov edi,dword ptr ss:[ebp+C] 004547EB 57 push edi 004547EC 6A 08 push 8 004547EE 53 push ebx 004547EF 8945 F8 mov dword ptr ss:[ebp-8],eax	004547E5	50	push eax
004547EB 57 push edi 004547EC 6A 08 push 8 004547EE 53 push ebx 004547EF 8945 F8 mov dword ptr ss:[ebp-8],eax	004547E6	FFD7	call edi
004547EB 57 push edi 004547EC 6A 08 push 8 004547EE 53 push ebx 004547EF 8945 F8 mov dword ptr ss:[ebp-8],eax	004547E8	8B7D 0C	mov edi,dword ptr ss:[ebp+C]
004547EE 53 push ebx 004547EF 8945 F8 mov dword ptr ss:[ebp-8],eax	004547EB	57	
004547EF 8945 F8 mov dword ptr ss:[ebp-8],eax	004547EC	6A 08	push 8
	004547EE	53	push ebx
	004547EF	8945 F8	mov dword ptr ss:[ebp-8],eax
	004547F2	FFDO	

Fig. 5 - Decrypted shellcode #1 function at 0x4547A8.

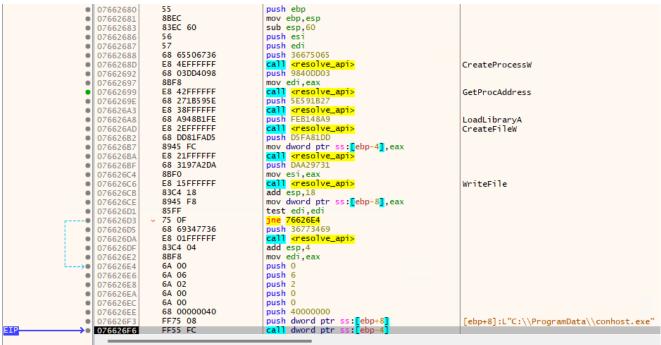
This function allocates memory on the heap and copies some data to it. The size of the region to be allocated is on the second argument. Going back to the main code, there's XOR decryption done on each piece of data, and subsequently a call to *VirtualProtect* to enable execution of the newly decrypted shellcode (Fig. 6). Their arguments are what seems to be a file path, a pointer to an executable, what seems to be an XOR key, and probably the size of the executable (0x12C00 bytes, or 76.8KB)

004515E6 004515E8 004515E8 004515E8 004515E8 004515E8 004515E2 004515F2 004515F5 004515F5 004515F5 004515F5 004515F5 004515F0 004515F0 004515F0 00451600 00451600 00451600 00451600 00451600 00451600 00451600 00451600 00451600 00451600 00451600 00451600 00451600 00451600 00451600 00451611 68 D0844E00 FF75 F4 00451611 0581610 FF70 FFD0	<pre>cal eax push dword ptr ss:[ebp-20] lea eax,dword ptr ds:[esi+A0] push colibri_04_09_2022.4530F8 push dword ptr ss:[ebp-C] push colibri_04_09_2022.4E84D0</pre>	edi:"MZE" VirtualProtect 4530F8:"18167" Febn-Cl:"MZE" 4E8400:L"C:\\ProgramData\\conhost.exe" shellcode #2					
dword ptr ss:[ebp-C]=[02D5FDB4 &"MZE"]=078ECA80 "MZE"							
.text:0045160E colibri_04_09_2022.exe:\$160E #A0E							
💷 Dump 1 🚛 Dump 2 💷 Dump 3 💷 Dum	np 4 📖 Dump 5 👹 Watch 1 📧 Locals 🤌	Struct 🕮 Disassembly					
Address Hex	ASCII						
078ECA80 4D 5A 90 00 03 00 00 04 00 00 0FF	FF 00 00 MZÿÿ						

Fig. 6 - Decrypted executable at 0x78ECA80.

After saving to file that executable and sending it to VirusTotal, we can see that it was <u>already uploaded</u>. Again, no luck on unpacking it with <u>unpac.me</u>. Entering the last decrypted shellcode, it seems to dynamically resolve some Windows API functions by passing a hash

and then it creates a file at C:\ProgramData\conhost.exe (Fig. 7).



dword ptr ss:[ebp-4]=[029CF610 <&CreateFileW>]=<kernelbase.CreateFileW>

Fig. 7 - Decrypted shellcode #2.

After running it, a file was indeed dropped at the expected location, which turns out to be the same file we manually dumped. VirusTotal shows <u>173 executables dropping this file</u>, most with compilation and first-seen timestamps from September 2022.

Let's have a look at the dropped file. Opening again on IDA Pro, looking at the main function, it seems very similar to the previous stage, almost a copy, with some minor changes. In the end, we can quickly spot the same pattern of resolving *VirtualProtect*, calling it, and then calling the decrypted shellcode, just as seen before. After running on x32dbg with a breakpoint at that last call, we can see as before that the pointer to the newly decrypted executable is the second argument (Fig. 8). However, this time the size is not being passed as an argument, but we can get it from other places, such as the call to a function that decrypts the executable, where the size to be decrypted is passed on the first argument.

00FF1565 00FF1568 00FF1569 00FF156C 00FF156C 00FF1574 00FF1574 00FF1577 00FF1577 00FF1577 00FF1578 00FF1578 00FF1581 00FF1584 00FF1588 00FF1588	8D45 D4 50 8B47 24 6A 40 FF75 DC FF75 E4 8D0458 0F870C30 8B47 1C 8D0488 8B0430 03C6 FFD0 68 C4480001 68 C4490001	<pre>lea eax,dword ptr ss:[ebp-2C] push eax mov eax,dword ptr ds:[edi+24] push 40 push dword ptr ss:[ebp-24] push dword ptr ss:[ebp-1C] lea eax,dword ptr ds:[eax+esi] mov zecx,word ptr ds:[eai+esi] mov eax,dword ptr ds:[edi+1C] lea eax,dword ptr ds:[eax+esi] add eax,esi call eax push conhost.10048C4 push conhost.1004920</pre>	edi+24:"OA\x0E" esi:"MZE" VirtualProtect 10048C4:"NtUnmapViewOfSction" 1004920:"ntdll.dll"		
 00FF1592 00FF1595 00FF1595 00FF1598 00FF159C 	FF7 <u>5</u> F8 8D85C4FEFFFF 50 8B45E4	<pre>push dword ptr ss:[ebp-8] lea eax,dword ptr ss:[ebp-13C] push eax mov eax,dword ptr ss:[ebp-1C]</pre>			
EIP 00FF159F	05 A0000000 FFD0	add eax,A0 <mark>call</mark> eax	shellcode		
dword ptr ss:[ebp-8]=[00EEF988]=07DCC838 .text:00FF1592 conhost.exe:\$1592 #992					
💭 Dump 2 💭 Dump 1 💭 Dum	np 3 💷 Dump 4	🕮 Dump 5 👹 Watch 1 🛛 💷 Locals	🖉 Struct		
Address Hex 07DCC838 4D 5A 00 00 01 00 00 00	2 00 00 00 FF FF 00	ASCII			

Fig. 8 - Decrypted executable at 0x7DCC838.

Going back, by putting a breakpoint on the call to decrypt the executable, we can see that the size is *0x5000* bytes (or 20KB). We can now extract the executable from memory and have a look at it. The file seems to be a valid executable with only two sections, *.text* and *.reloc*. There's sufficient evidence to conclude that we have successfully unpacked the ColibriLoader.

At the time of this research, this last stage was not yet on VirusTotal. Later, we found a quicker way of unpacking the malware. We used <u>this script</u> to extract executables from memory dumps (from a sandbox run for example) and then the YARA rule we share below was used to find the Colibri sample.

Deobfuscating ColibriLoader

Finally, let's have a very quick look at the actual malware. The first anti-analysis trick we encounter is called *opaque predicates* (Fig. 9); It's a commonly used technique in program obfuscation, intended to add complexity to the control flow. There are many patterns of this technique but in this case, the malware author simply takes an absolute jump (JMP) and transforms it into two conditional jumps, jump if zero (JZ) and jump if not zero (JNZ). Depending on the value of the Zero flag (ZF), the execution will follow the first or second branch. However, disassemblers are tricked into thinking that there is a fall-through branch if the second jump is not taken (which is impossible as one of them must be taken) and try to disassemble the unreachable instructions (often invalid) resulting in garbage code.

.text:00405623 ;		
.text:00405623		
.text:00405623	public	start
.text:00405623 start:		
.text:00405623	push	ebx
.text:00405624	push	esi
.text:00405625	push	edi
.text:00405626	jz	short near ptr loc_40562A+1
.text:00405628	jnz	short near ptr loc_40562A+1
.text:0040562A		
.text:0040562A loc_40562A:		; CODE XREF: .text:00405626↑j
.text:0040562A		; .text:00405628↑j
.text:0040562A	mov	eax, 0FFFF4DE8h
.text:0040562F	call	fword ptr [edi+5Eh]
.text:00405632	рор	ebx
.text:00405633	retn	
.text:00405634 ;		

Fig. 9 - Example of ColibriLoader opaque predicates anti-analysis technique. In order for IDA Pro to load it properly, we need to patch the first conditional jump to an absolute jump and *NOP* out the second jump (Fig. 10). We've automated this task using <u>myrtus0x0</u>'s code since SmokeLoader also uses this technique.

	;	== S U B	ROUTINE =			
.text:00405623						
.text:00405623						
.text:00405623	; int start()					
.text:00405623		public s	start			
.text:00405623	start	proc nea	ar			
.text:00405623		push	ebx			
.text:00405624		push	esi			
.text:00405625		push	edi			
.text:00405626		jmp	short loc_40562	3		
.text:00405626	;					
.text:00405628		db 3 dup	o(90h)			
.text:0040562B	;					
.text:0040562B						
.text:0040562B	loc_40562B:			; CODE	XREF:	start+3↑j
.text:0040562B		call	sub_40557D			
.text:00405630		рор	edi			
.text:00405631		рор	esi			
.text:00405632		рор	ebx			
.text:00405633		retn				
.text:00405633	start	endp				

Fig. 10 - Patched ColibriLoader.

The last analysis we did was trying to extract the strings the malware uses, which will contain indicators of compromise, such as command and control servers. After looking a bit through the code, it wasn't hard to find the string decryption function at *0x40594B* since there are 71 cross-references for it, so it's probably the most used function (Fig. 11 and 12).

Fig. 11 - Example call to the string decryption function.

void __cdecl sub_40594B(WORD *str_enc, unsigned int str_len, WORD *key, unsigned int key_len)

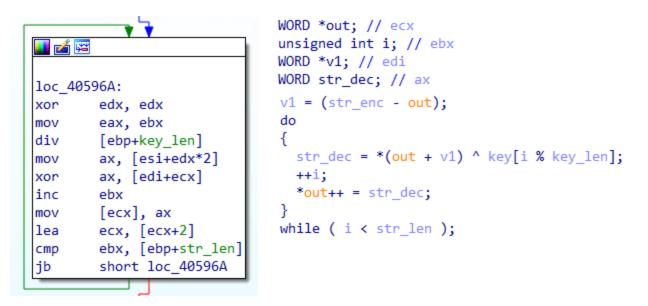


Fig. 12 - String decryption loop from function at 0x40594B.

This code seems straightforward enough. Strings are encrypted with an XOR key passed as an argument to the function. Yet, we didn't need to script this out because we've found a working <u>IDA script</u> from Casperinous. Here are the results:

0x401faf %s\\%s 0x401fd4 \\Microsoft\\WindowsApps 0x401ffa Get-Variable.exe 0x402020 powershell.exe -windowstyle hidden 0x402046 %s:Zone.Identifier 0x40229c %s\\%s 0x4022c1 \\WindowsPowerShell 0x4022e7 dllhost.exe 0x40230d %s:Zone.Identifier 0x402579 %s\\%s 0x40259e \\Microsoft\\WindowsApps 0x4025c4 Get-Variable.exe 0x402706 %s\\%s 0x40272b \\WindowsPowerShell 0x402751 dllhost.exe 0x4028e5 %s:Zone.Identifier 0x402999 runas 0x4029bf cmd.exe 0x4029e5 /c %s%s%s %s

0x402b49 %s\\rundll32.exe %s,%s 0x402c4e %s /s 0x402c74 runas 0x402ca8 %s\\System32\\regsrv32.exe 0x402cff %s\\SysWOW64\\regsrv32.exe 0x402e48 /c %s%s%s %s 0x402e6e cmd.exe 0x402e94 open 0x403041 %s%s 0x403612 6rmUi1hRdfbV0QyXqAoT 0x4037c0 /c chcp 65001 && ping 127.0.0.1 && DEL /F /S /Q /A %s%s%s 0x4037e5 cmd.exe 0x4038db Software\\Microsoft\\Windows NT\\CurrentVersion 0x403903 ProductName 0x4039cc Unknown 0x403c07 %08IX%04IX%lu 0x403c81 /create /tn COMSurrogate /st 00:00 /du 9999:59 /sc once /ri 1 /f /tr 0x403ca7 %s\\schtasks.exe 0x403e4b %s\\schtasks.exe 0x403e71 /delete /tn COMSurrogate /f 0x404058 Content-Type: application/x-www-form-urlencoded 0x404488 zpltcmgodhvvedxtfcygvbgjkvgvcguygytfigj.cc 0x4044ad yugyuvyugguitgyuigtfyutdtoghghbbgyv.cx 0x404582 /gate.php 0x4045a8 hf9gkeO66MP7WJXkg9rp 0x4045ce 2OrnJZG6Wtbzd4bKJoS0 0x4045f4 %s?type=%s&uid=%s 0x40461a check 0x404640 GET 0x404666 HTTP/1.1 0x40489d 1.2.0 0x4048c3 Build1 0x4048e9 /gate.php 0x40490f hf9qkeO66MP7WJXkg9rp 0x404934 2OrnJZG6Wtbzd4bKJoS0 0x40495a %s?type=%s&uid=%s 0x404980 update 0x4049a6 POST 0x4049cc HTTP/1.1 0x4049f2 %s|%s|%s|%s|%s|%s 0x404a18 32bit 0x404a3e 64bit

0x404d9c Build1 0x404dc3 /gate.php 0x404dec hf9qkeO66MP7WJXkg9rp 0x404e13 2OrnJZG6Wtbzd4bKJoS0 0x404e3a %s?type=%s&uid=%s 0x404e61 ping 0x404e61 ping 0x404e88 POST 0x404eaf HTTP/1.1 0x404eaf HTTP/1.1 0x404ed6 %s|%s|%s|%s|%s|%s|%s 0x4054d3 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

These decrypted strings also allow us to further reverse the malware more quickly if we need to.

Wrap-up

We presented a way to manually unpack the ColibriLoader samples from a campaign linked to the threat actor <u>UAC-0113</u>. Later, we found a quicker way of unpacking the malware using the <u>pe_extract.py</u> script combined with a YARA rule which detects unpacked samples of ColibriLoader, which we share below. All indicators of compromise and threat-hunting rules can be found at <u>https://github.com/bitsight-research/threat_research</u>

Threat Hunting Signatures

Here's a **YARA rule** to detect packed ColibriLoader samples based on a typo:

The following **YARA rule** detects unpacked ColibriLoader samples based on the string decryption function. This rule was tested on VirusTotal and it returned few results with first-seen timestamps between September 2021 and November 2022.

Here's a **Suricata rule** to detect the ColibriLoader network traffic, specifically its C2 check-in request, tested with a PCAP generated from a <u>sandbox run</u> of the malware:

Indicators of Compromise

Unpacked ColibriLoader sample - 59f5e517dc05a83d35f11c6682934497

173 Packed ColibriLoader samples: <u>https://github.com/bitsight-</u> <u>research/threat_research/blob/main/colibriloader/packed_colibri_samples.txt</u>

More at <u>https://github.com/bitsight-research/threat_research</u>